Development of microcontroller based over-current relay controls

A Thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology in Electrical Engineering

By

SAMBIT SENAPATI (110EE0228)

A.V.JAYA KRISHNA (110EE0424)

KUMAR ROHIT (110EE0560)

Under supervision of
Prof. Supratim Gupta

Department of Electrical Engineering
National Institute of Technology, Rourkela
May 2014

Department of Electrical Engineering,
NIT Rourkela,
Odisha, India - 769008

Certificate

This is to certify that the work in this project entitled **"Development of microcontroller based over-current relay controls"** by **Sambit Kumar Senapati (110ee0228), A. V. Jay Krishna (110ee0424)** and **Kumar Rohit (110ee0560)** has been carried out under my supervision in partial fulfillment of the requirements for the degree of Bachelor in Technology during session 2013-14 in the department of Electrical Engineering, National Institute of Technology Rourkela, and this work has not been submitted elsewhere for a degree.

The candidates have fulfilled all the prescribed requirements.

In my opinion, bachelor of technology degree in Electrical Engineering may be awarded based on this thesis.

Place: Rourkela
Dept. of Electrical Engineering
National institute of Technology
Rourkela-769008

Prof. Supratim Gupta

ACKNOWLEDGEMENTS

We would like to articulate our deep gratitude to our project guide Prof. SUPRATIM GUPTA who has always been source of motivation and firm support for carrying out the project. We would also like to convey our sincerest gratitude and indebtedness to all other faculty members and staff of Department of Electrical Engineering, NIT Rourkela, who showed their great efforts and guidance at required times without which it would have been very difficult to carry out our project work. Moreover, an assemblage of this nature could never have been attempted with our reference to the works of others whose details are mentioned in the references section at the last. We acknowledge our indebtedness to all of them. Furthermore, we would like to take the name of our parents and God who directly or indirectly encouraged and motivated us during this dissertation**.**

Sambit Kumar Senapati

A. V. Jaya Krishna

Kumar Rohit

ABSTRACT

This Project is about designing the controls of an Overcurrent relay using microcontroller. Relay is an electrical switch. Generally it operates on electromagnetic principle for switching action. Over current (OC) relay is a type of relay which performs its switching action based on an input current to it. When the input current exceeds a particular pick up value the relay operates.

Microcontroller based relays are gaining popularity as these have no mechanical contacts. These can operate at very high speeds. Any kind of characteristics can be obtained by using these. In a typical microcontroller based relay there is a current transducer, a controller, current transformer and a circuit breaker.

Microcontroller reads load current via CT, transducer and ADC. When current is above pickup it gives a signal to circuit breaker to trip. Microcontroller can be programmed to obtain IDMT characteristics.

Table of Contents

LIST OF FIGURES:

Chapter I

Introduction

1. Introduction:

Relay is an electrical switch. Generally it operates on electromagnetic principle for switching action. Relays used in power system for equipment protection are called as protective relays. Over the time, relays have evolved. Now a days microcontroller based relays are used for more sophisticated switching action.

Over current (OC) relay is a type of relay which performs its switching action based on an input current to it. When the input current exceeds a particular pick up value the relay operates. Previously OC relays were electromagnetic type.

OC relays have two settings:

1. Time setting multiplier: Operational time delay of OC relay will be determined by this.

2. Plug setting multiplier: The PSM decides the pickup current required for the OC relay.

This Project will attempt to design various controls of OC relay using microcontroller. Microcontroller will issue a trip signal when the input current will be higher than pickup current. Microcontroller to be used is ATMEGA32 (ATMEL AVR series).

Steps included:

- Current to voltage conversion.
- Measurement of RMS value of load current via ADC
- 16x2 alphanumeric LCD interfacing for display.
- Developing algorithms for various IDMT curves.
- Simulation and Hardware implementation.

1.1 Objectives:

To design and fabricate the various controls of OC relay using ATMEGA32 and developing algorithms for various IDMT curves.

Chapter II

Background and Literature Review

[9]

2. Background and Literature Review

This chapter will discuss significant parts of protection systems, OC relays and there importance.

Also it includes a section on ATMEGA 32.

2.1 Need for Protection

Fault imposes serious threat on both equipment and user. Therefore in power system there is every need of protection. Without it power system will not sustain few hours.

Various protection areas are:

- User/personnel safety
- Equipment safety: Safeguard equipment from various abnormal and faulty conditions
- General Safety: Prevent power system from secondary accidents such as fire
- Power system stability: Maintaining a continuous and reliable power supply

2.2 Overcurrent Relay

Over current as defined by the National Electrical Code is any current which is greater than the current rating of equipment or the ampere capacity of a conductor. Overload, short circuit, or ground fault may be some of its reason. When current flows through a conductor it produces some amount of heat and this heat is proportional to current flowing in conductor directly. Unwanted heat may damage the equipment. So current should not go over the equipment's current rating. So overcurrent protective devices are used to protect circuits or equipment from overcurrent and hence short-circuit and overloads.

Over-current (OC) relay takes input from current transformer (CT) in the form of ac current. In normal condition relay outputs open state which is changed to closed state in abnormal or overcurrent situation. The relay is consists of two basic settings: Time setting multiplier and the plug setting multiplier. The time setting decides Operational time delay of OC relay while the plug setting decides the pick-up current of relay.

OC relay may be of any of the following type:

A. **Instantaneous OC relay:** Instantaneous actually means no intentional time delay. The operational time of an instantaneous OC relay is very small. So such a relay has only PSM but no TSM
B. **Definite Time OC relay:** A definite time OC relay is that which has an intentional time delay, after which the relay issues trip signal to circuit breaker (CB). Thus, it has both TSM and PSM.
C. **Inverse Time OC relay:** Inverse time relay is very useful. It operates quickly during large current and gives a time delay during small current flow. Electromechanical relay has this characteristic natural to it. Now with microcontroller and microprocessor with us, it is possible to get any shape of characteristic we want. This has led to 3 important Inverse time characteristics:
    - **Inverse Definite Minimum Time (IDMT) OC relay:** This is possibly the most widely used characteristic. The curve is inverse for low current values, which turns to constant value i.e. gives a minimum time delay at higher values of current. This happens due to the fact that for large currents the flux saturates in the electromechanical relays and also the operating torque. This torque is proportional to the square of the flux.

      $T_{delay} = 0.14 \times (TMS) / ((PSM)^{0.02}-1)$
    - **Very inverse time overcurrent relay:** The inverseness of this curve is higher than that of the IDMT curve.

      $T_{delay} = 13.5 \times (TMS) / (PSM-1)$
    - **Extremely inverse time OC relay:** The inverseness of this characteristic is higher than that of the very inverse time OC relay characteristic.

      $T_{delay} = 80 \times (TMS) / (PSM^2-1)$

T_{delay}= time of operation
TMS= Time multiplier setting
$PSM = I/I_s$
I=load current
I_s=Pick up current

2.3ATMEGA32 microcontroller:

ATMEGA32 is an AVR series microcontroller made by the ATMEL Corporation. This is enhanced RISC architecture made 8-bit microcontroller which consumes very low power. It executes long instructions in single clock cycle. Hence its throughputs are about 1 MIPS per MHz. This allows the system an optimization of consumption of power at different speed.

ATMEGA32 is now days widely used by hobbyist due its user friendly programming environment and low cost. It has got many in-chip features which makes it more of a choice rather than other microcontrollers. It has got 4 I/O ports and In-chip ADC channels which makes suitable to this project.

Now days it is being widely used in robotics application. Many college competitors use it for line follower or for other embedded projects. By reducing cost and power consumption it has certainly got an edge over other microcontrollers.

Various features:

- AVR 8-bit Microcontroller with high performance and low power consumption
  - Based on enhanced RISC Architecture
  - 131 number of useful instructions (Mostly Single-clock Cycle Execution)
  - General Purpose Working Registers – 32 x 8
  - Throughput achieved is 16 MIPS at 16 MHz
  - Multiplier on chip with 2 cycle execution time
- Memory segments with high longevity
  - 32K Bytes Flash program memory of In-System Self-programmable type
  - EEPROM is of 1024 Bytes
  - Internal SRAM of 2KB
  - Write or Erase Cycles: 10,000 for Flash and 100,000 for EEPROM
  - Program lock feature is there for software security
- JTAG Interface(IEEE standard 1149.1)
  - Capabilities of boundary scan according to the JTAG Standard
  - Support of on chip debug is extensive
  - Through the JTAG Interface you can program the Flash, EEPROM, Fuses, and Lock Bits
- Peripheral Features
  - Two number of 8-bit Timers (Counters) - Separate Prescaler setting and Comparison Modes
  - One 16-bit Timer (Counter) - Separate Prescaler, Comparison Mode and Capture Mode
  - Counter is provided with separate oscillator
  - PWM usage using 4 channels
  - 8 number of 10-bit ADCs (can be used as 8-bit)
  - 8 number of single channels
  - 7 Differential Channels is provided in TQFP package
  - Programmable Gain can be 1x, 10x, or 200x for diff. channels
  - Two-wire Serial Interface which is Byte oriented.
  - Serial USART - programmable
  - SPI Serial Interface (master-slave feature)
  - Watchdog-Timer (programmable) with Separate oscillator
  - Analog Comparator is on the chip
- Special Microcontroller Features
  - Power on Reset and Programmable Brown out Detection

- RC Oscillator is internally calibrated
- Both external and internal Interrupt available
- Six number of sleep Modes which are Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby

➢ Input/Output and Packages
- I/O Lines – 32 number (programmable)
- 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF

➢ Operational Voltage level
- 2.7-5.5V for ATmega32L
- 4.5-5.5V for ATmega32

➢ Power Consumption at 1 MHz, 3V, 25°C for ATmega32L [7]
- Active state - 1.1 mA
- Idle Mode - 0.35 mA
- Power down Mode - < 1 micro Ampere

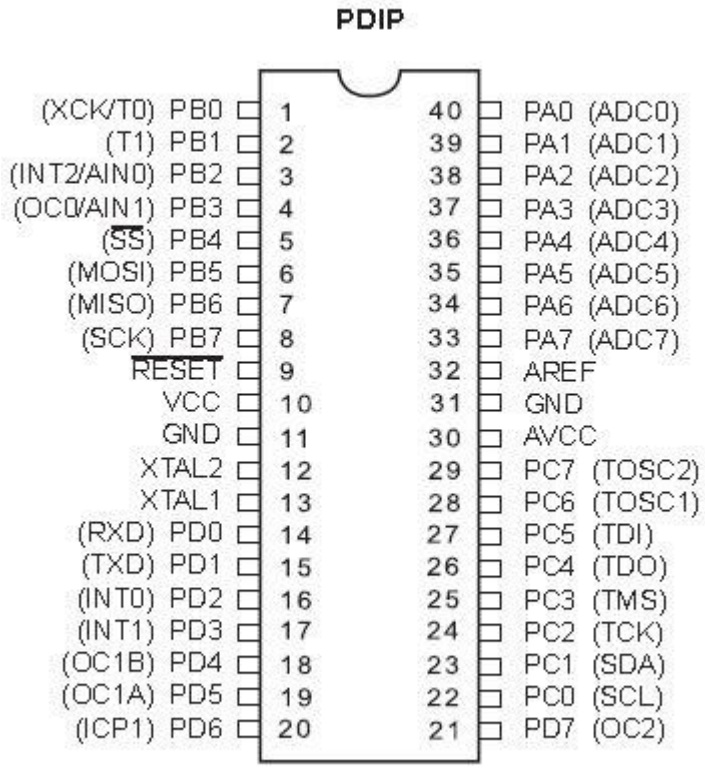


Fig.1 ATMEGA32 Pin Diagram [7]

Chapter III

Methodology

3. Methodology

This chapter presents methodology used to design the OC relay including hardware and simulation in PROTEUS environment. It will include Atmega32 ADC utilisation, interfacing of 16x2 alphanumeric LCD to ATMEGA32 circuit and ATMEGA32 assembler programming. Also some algorithms are explained.

Brief review of system is necessary before designing it or simulating it. Figure below represents the complete system design of OC relay.

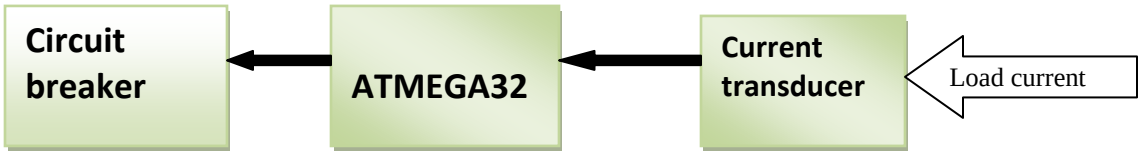


Fig.2 OC relay block diagram

The main idea in this work is to trip circuit breaker during over-current with the help of ATMEGA32. During faulty condition this overcurrent will be sensed by CT and it will be converted to voltage by use of current sensors or standard resistances.

Suppose current is 'I' and resistance used is 'R' then voltage will be 'IxR'. Since voltage is directly related with current we will be calculating RMS value of voltage rather than current as our microcontroller only takes voltage as input. Now this voltage will be sampled with help of ADC and RMS is calculated.

3.1 ADC interfacing

ADC with sensor is like senses to microcontroller; it can sense the surrounding such as rotation, sound, gravity, acceleration, smell, distance etc. Inside AVR there is an ADC port. The PORTA is multiplexed with ADC i.e. the PORTA pins are shared with ADC channels. The ADC has two modes; single conversion and free running mode. ADC does conversion only one time in single conversion mode. In the free mode it does the conversion repeatedly.

- **ADC Prescaler**: A clock pulse is required for ADC to do its conversion. ADC requires lower range of frequency. Hence the system clock is divided to lower range. Frequency should lie between 50-200 kHz. For accuracy we go for lower frequency range but if conversion is required to be completed quickly then higher range of frequency is selected. This frequency can be set to intended value by user internally. Here comes the concept of **prescaling.** System clock is prescaled by 2, 4, 16, 32, 64, and 128 through programming i.e. putting some value into the required register.

- There are 8 number of ADC in ATMEGA32. So we can choose either of the channels for conversion. We can also use all the channels at a time but this project does not require it.

The ADC has only four registers.

1. **ADC Multiplexer Selection Register (ADMUX)**: This used to select the required channel and the reference voltage level.
2. **ADC Control and Status register A (ADCSRA)**: It gives the status of conversion and also used for controlling ADC.
3. **The ADC Data Register – ADCL and ADCH**: ADC result is obtained at this register.

ADC Codes Explained:

For codes refer to appendix A

Each conversion in AVR takes about 1/125k seconds (as the prescaler is set at 128 and clock frequency is 16 MHz. We are using ADC in manual mode. After each conversion we are triggering for next conversion.

8-bit ADC result is obtained in ADCH register. Now we have to convert this to BCD format. We are extracting the result correct up to 1 decimal place. So if result is 4.9V then it is stored as 49.

We are doing a trick here to do this fast. Since result is 8-bit its max value is 256. And we are representing this max value as 50. So let our required result is 'A'. Then this is equal to 50xADCH/255. Since 255 is just 1 less than 256 we are dividing by 256. So for accuracy we are multiplying in above with 51 instead of 50. So now 'A' is equal to 51xADCH/256. Multiplying in AVR is done using MUL command. But there is no division command in AVR. But 256 = 2^8. So dividing by 256 is nothing but shifting our result 8 times to right or we can just ignore low byte of result.

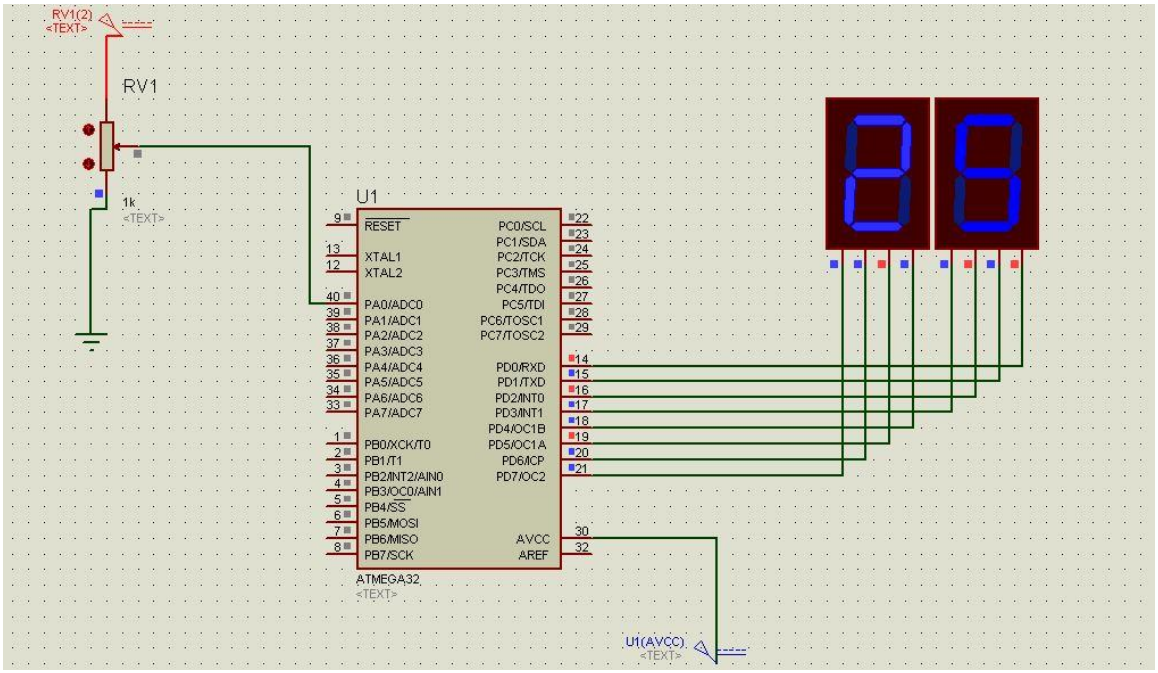


Fig.3 results of ADC at input 2.5 V

3.2 Integer Square Root Algorithm:

This can be done by simple math trick. We know that sum of odd numbers from 1 to (2n-1) is n^2 i.e. $1+3=4=2^2$, $1+3+5=9=3^2$, etc. so let take a number 27 its integer square root is 5. We can obtain this by subtracting consecutive odd numbers until subtraction yields negative result. So by counting the number of subtraction we can find the result.

Algorithm

Count=0
N=27; number whose square root to be obtained
S=1; this is first number to be subtracted
N=N-S;
If (N-S>0)
    Count=Count+1;
Else break;
For codes refer to Appendix A

But for bigger numbers this not a good method as number cycles increases linearly. So a fast square root algorithm is required.

3.3 LCD interfacing:

We can know from the name itself that a 16x2 LCD has 16 pins. It has pins for input voltage, ground, contrast, data-pins, control pins and back light selection pins.
Pin diagram of 16x2 LCD used is shown in following figure.

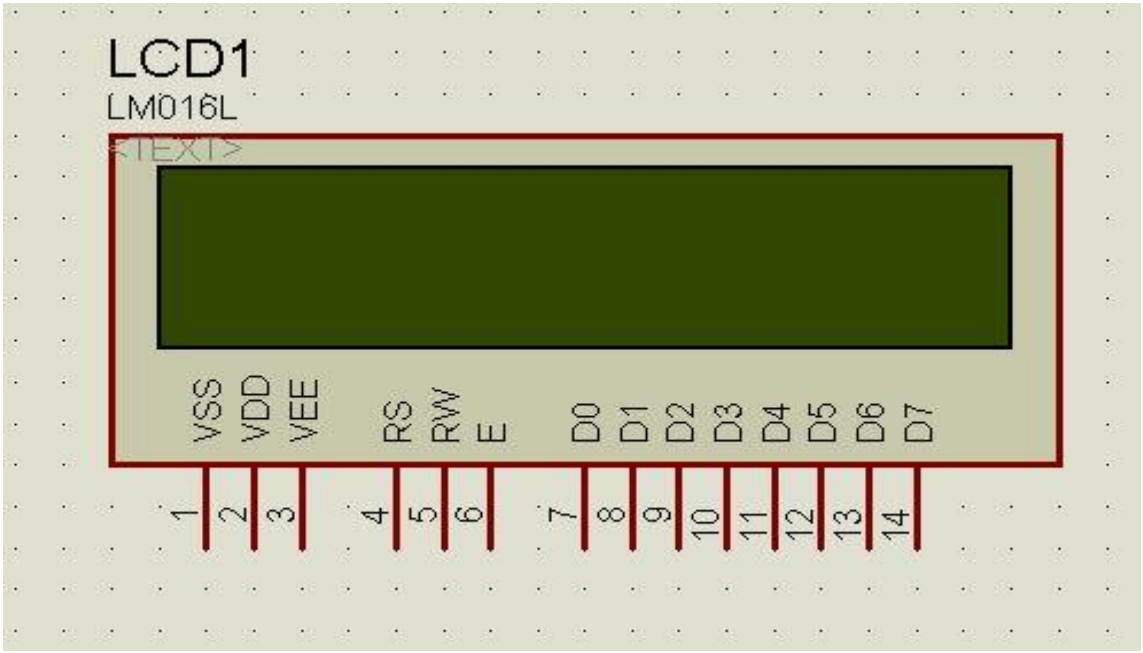


Fig.4 LCD Pin Details

VSS: This is power pin of LCD.
VEE: This determines the contrast level. For 5V it gives brightest contrast and for 0V it gives minimum contrast. Contrast of LCD can be controlled by varying voltage between 0-5V at this pin.
RS: This is one of the control pin which is the register select pin. This is set to logic 1 for writing data to LCD. But in command mode this has to be set to logic 0 initially.
R/W: This is the read-write pin of LCD. While writing something to LCD it is set to low value but if something is to be read from LCD this has to be set to high value.
E: This is the Enable pin. This has to toggle during writing mode or reading mode because it acts as a clock pulse to ADC.

Steps for write mode are:

- RW set to logic 0
- RS set in data mode
- E is set to logic 1
- Data is put in the bus
- Give some time delay
- E set to logic 0

3.4 Algorithm steps for OC relay:

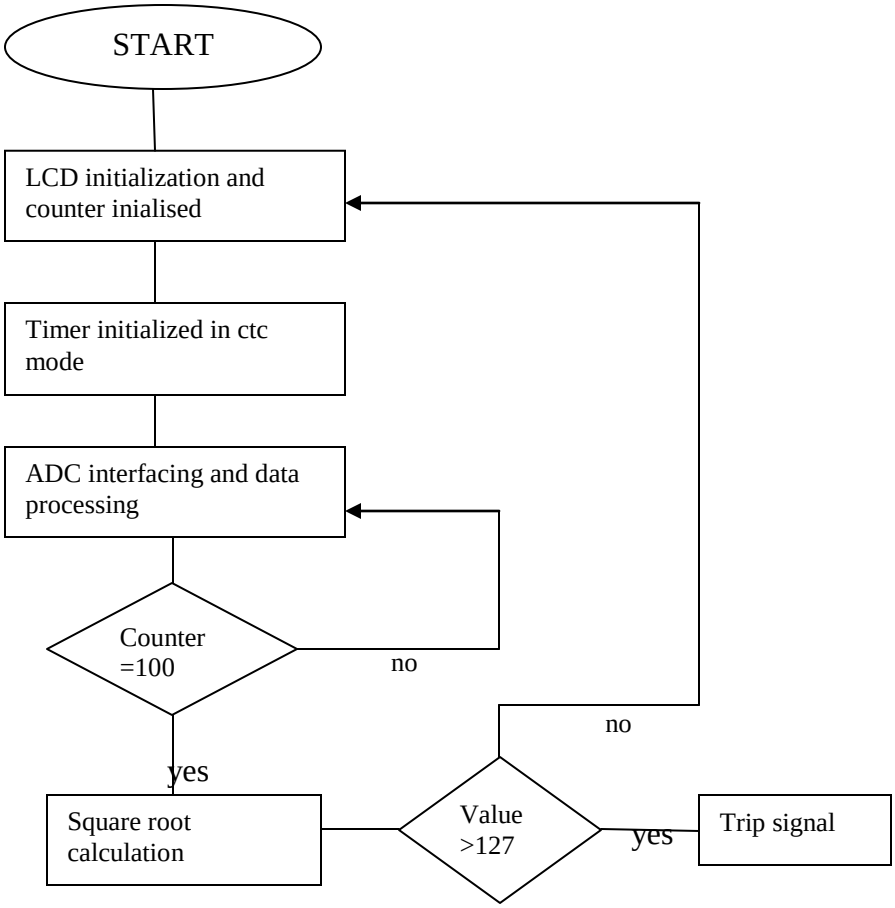
Step1: LCD is initialised.

Step2: Timer0 is initialised in CTC mode. OCCR0 value is set at 50 so that we can take 100 sample of sine voltage in a complete cycle. 1 count of timer takes 4us at 64 prescaler setting. So for 50 numbers of counts time taken will be 200us. This will give us 100 samples for 50 Hz frequency.

Step3: ADC is interfaced and voltage sample is taken via PA0 terminal. This data is now processed in Q3.5 format and square of this voltage is calculated and stored. Each sample voltage is squared and added to the sum. This process is repeated for 100 values. To keep track of it a counter is used.

Step4: when counter reach 100 then square root of total sum is calculated. This gives us the root mean square voltage. Now this square root value (int) is compared with 127. If it is higher than 127 then trip signal is issued.

Fig.5 Flowchart of Program



Results:

RMS voltage at different peak voltage was calculated assuming signal to be sine wave and of 50 Hz.

Some results are shown below:

For Vp = 5V, Vrms(actual)=3.54V, Vrms(calculated) = 3.6V, error = 1.67%

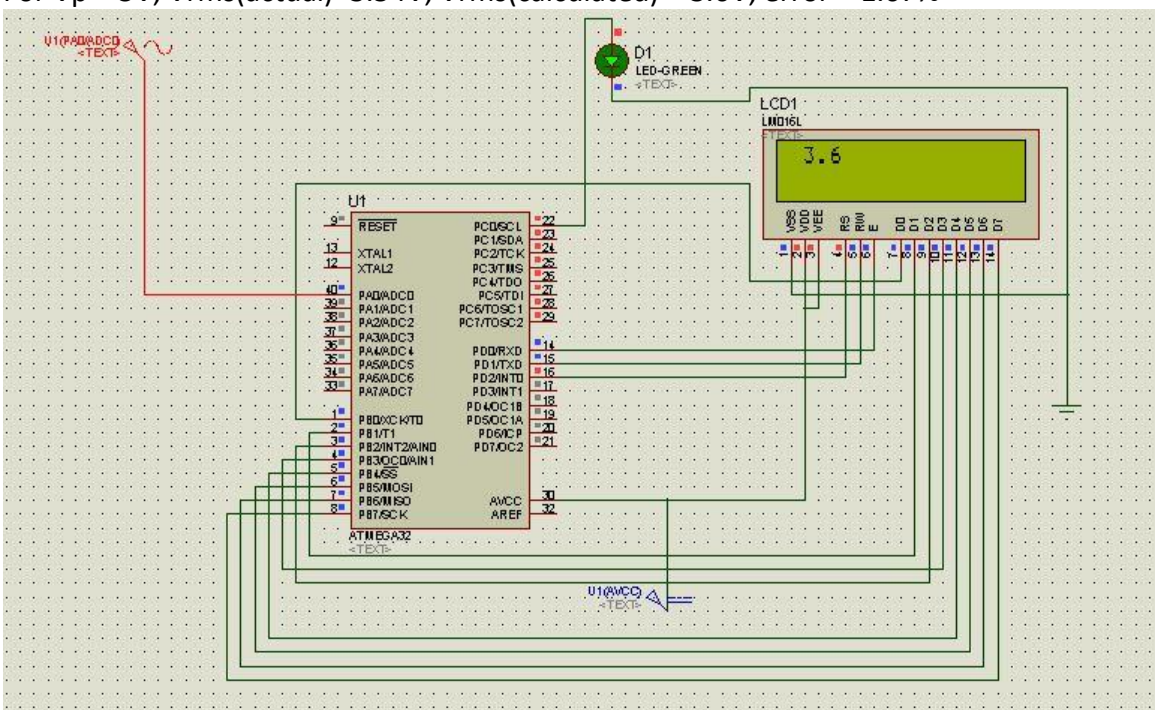


Fig.6 rms result at Vp=5v

For Vp=4V, Vrms actual = 2.83V, Vrms calculated = 2.8V, error = -1.1%

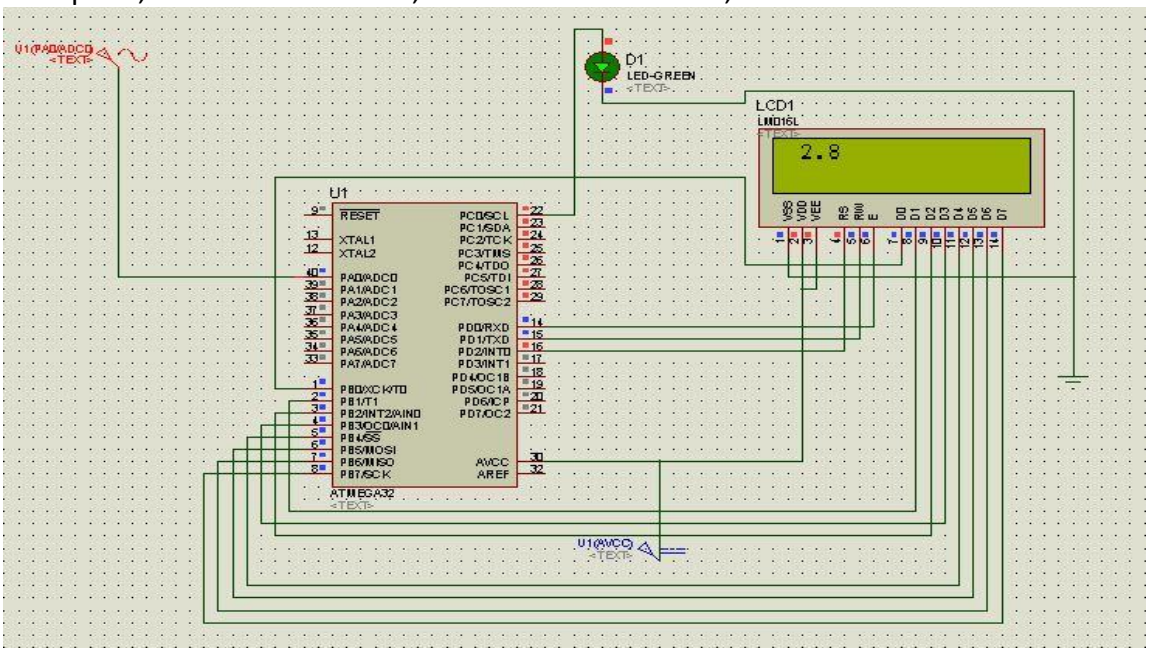


Fig.7 rms result at Vp=4V

For Vp=3V, Vrms actual = 2.12, Vrms calculated = 2.1V, error = -0.95 %

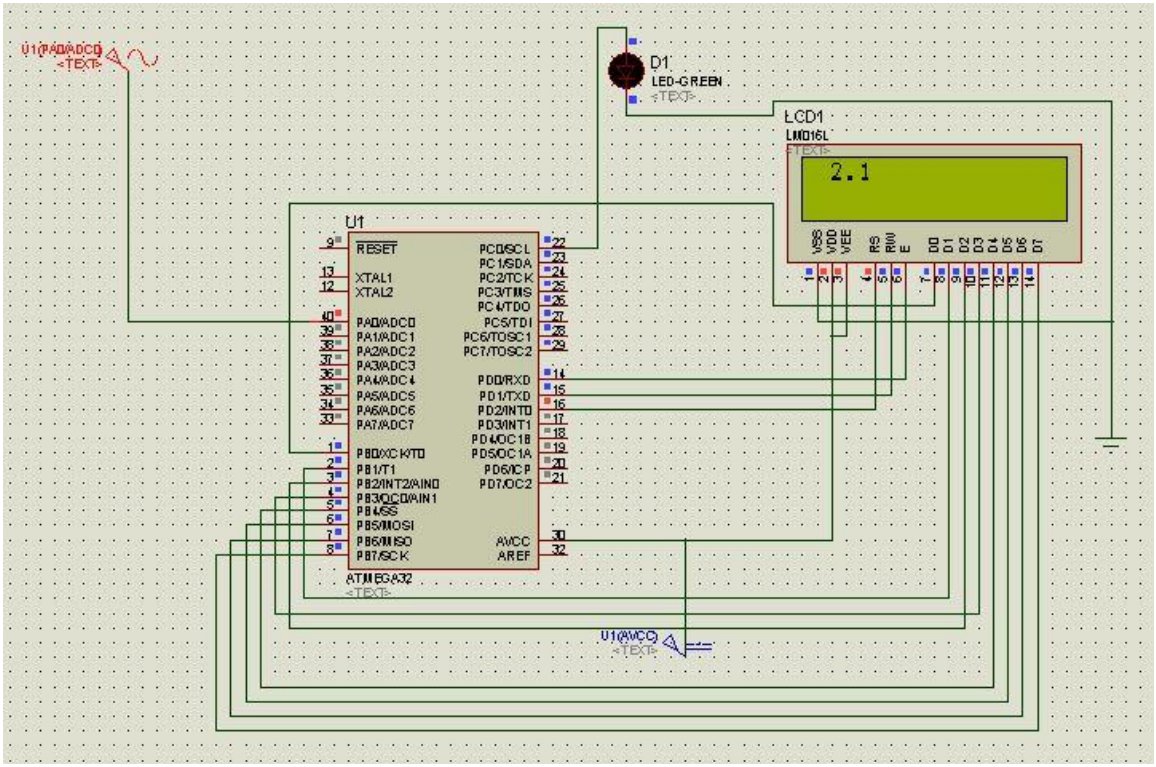


Fig.8 RMS result at Vp=5V

Conclusions:

All the results were satisfactory and rms voltage calculated with minimum error. And it was seen that for rms voltage below 2.5V trip signal is indicated by an LED connected at PortC. Error in calculation was found out to be within 5%.

REFERENCES

[1] Zoolnasri Bin Abu Harun, "OVER CURRENT PROTECTION RELAY USING PIC MICROCONTROLLER", University Malaysia Pahang, 2007

[2] M. Chander and B. Ravindranath, "Power System Protection and Switchgear", 2nd edition, NAI publication, 2011

[3] S. R. Bhide and Y. G. Paithankar, "Fundamentals of Power System Protection", 2nd edition, PHI, 2010

[4] Dhananjaya V. Gadre, "Programming and Customisation of AVR microcontroller", 2nd edition, TMH, 2001

[5] James Ulery, fast integer square root algorithm, University of Toronto, 2006

[6] B. Collis, "An Introduction to Practical Electronics, Microcontrollers and Software Design", 2nd edition, 2013

[7] ATMEGA32 Datasheet, Atmel Corporation, www.atmel.com

Appendix A

Assembly codes for ADC:

```
.include "m32def.inc"
.org 0x0000
.DEF ANSL = R0
.DEF ANSH = R1
rjmp main

.org $020  ;ADC completion ISR vector
rjmp ADC_ISR                    ;jump to adc interrupt subroutine

main:
ldi r16 , low (RAMEND)          ;initialising stack pointer
out SPL , r16                   ;needed for calling any subroutine
ldi r16 , high (RAMEND)         ;loading stackpointer to ram end address
out SPH , r16

ldi r16, 0xFF                   ;set portd as o/p port
out DDRD, r16

sei

sbr r16, 1<<ADPS2  + 1<<ADPS1 + 1<<ADPS0 + 1<<ADIE + 1<<ADEN

;prescaler - 16MHz/128
;1<<5 for free running mode, ADIE - adc intrpt enable
;ADEN- adc enable

sbr r17, (1<<ADLAR)+(1<<REFS0);set reference to AVcc by REFS0 and

;get a 8bit result on ADCH by left adjusting via ADLAR

out ADMUX, r17
sbr r16, (1<<ADSC)              ;start conversion
out ADCSRA, r16
loop:
rjmp loop

ADC_ISR:
push r16
in r16, ADCH                    ; get i/p from ADC conversion
rcall ADC_result_convert
out PortD,ANSH
pop r16
out ADCSRA,r16
reti ; return from interrupt

;************************************************;
```

```
ADC_result_convert:
;5.0 is stored as 50
;so 50*ADCH/256 is aur ans
;since ADCH max value is 255 and we are dividing by 256 so multiplication is 51
;instead of 50
ldi r17,51
mul r16,r17
;the result of mul is stored in r1:r0
;divide by 256 = 2^8 means shifting 8 times to the right
;otherwise we can just ignore r0 register
;result in r1 register
rcall BCD_convert
ret
;****************************************************;

BCD_convert:
mov r17,r1
push r1
rcall divide10
ldi r17,10
mul r17,r18
mov r16,r0
pop r1
sub r1,r16
LSL r18
lsl r18
lsl r18
lsl r18
or r1,r18
ret

divide10:
;************************************************************;
;loop execute divide by 10
;1/5 x 255 = 255 / 5 = 51 and 255 nearly equal to 256
;since we are dividing by 1
;A / 10 ~= [ [ ( A x 51 ) + 1 ] / 2 ] / 256
;256 x [A / 10] ~= ( 51A + 1) / 2
;256 x [A / 10] ~= 51A/2 + 1/2
;256 x [A / 10] ~= 51A/2
;256 x [A / 10] ~= 25.5 x A
;256 x [A / 10] ~= 25A
;since we dropped 1/2 earlier so we are multiplying by 26 instead of 25
;so  256x[A/10] ~=26A
;so we multiply with 26 and ignore lower byte of result
ldi r16,26
mul r16,r17
mov r18,r1
```

```
ret
```

Square root

```
.DEF  ANS = R0          ;To hold answer
.DEF    A = R16         ;To hold the square
.DEF    B = R18         ;Sum, Work space

 LDI A,100              ;Load the square into A

SQRT:
LOOP:
SUB A,B                 ;Subtract B from Square
BRCS DONE               ;If bigger than sqaure we are done
INC ANS                 ;Increment the answer
SUBI B,-2               ;Increment B by two
RJMP LOOP
DONE: RJMP DONE
```

16 bit fast square root code

```
;-------------------------------------------------------------------------------:
; 16bit square root
.include"m32def.inc"
.DEF      var0 = R17
.DEF      var1 = R18
.DEF      var2 = R19
.DEF      var3 = R20
.DEF      var4 = R21
.DEF      var5 = R22
.DEF      var6 = R23
; var[6:2] = <work area>      .
;
; Result:var[0]   = Result (8bit)
.org 0x0000
ldi r17 , low (RAMEND)              ;initialising stack pointer
out SPL , r17                       ;needed for calling any subroutine
ldi r17 , high (RAMEND)             ;loading stack pointer to ram end address
out SPH, r16
ldi var1,0x09
ldi var0,0xb6

sqrt16:             clr       var2
                    clr       var3
                    ldi       var4,1
```

```
                clr     var5
                ldi     var6,8
sqrt16l: lsl    var0
                rol     var1
                rol     var2
                rol     var3
                lsl     var0
                rol     var1
                rol     var2
                rol     var3
                brpl    PC+4
                add     var2,var4
                adc     var3,var5
                rjmp    PC+3
                sub     var2,var4
                sbc     var3,var5
                lsl     var4
                rol     var5
                andi    var4,0b11111000
                ori     var4,0b00000101
                sbrc    var3,7
                subi    var4,2
                dec     var6
                brne    sqrt16l
                lsr     var5
                ror     var4
                lsr     var5
                ror     var4
                mov     var0, var4
                ret
```

Full Program

```
.include "m32def.inc"
.DEF  ANSL = R0                    ;Square Root (answer)
.DEF  ANSH = R1                    ;
.DEF  REML = R2                    ;Remainder
.DEF  REMH = R3                    ;
.DEF    AL = R19                   ;Square to take root (input)
.DEF    AH = R20                   ;
.DEF     C = R21                   ;Loop Counter
.def reg17 = R17

.def uni = R16 //universal register for all data transfers and interim storage of values

.equ L_RS = 2  //RS connected to PD.0
.equ L_RW = 1 //RW connected to PD.1
.equ L_E = 0 //E connected to PD.2
.org 0x0000
rjmp main
.org $016
rjmp timer0_ISR
.org $020                          ;ADC copletion ISR vector
rjmp ADC_ISR                       ;jump to adc iterrupt subroutine

main:
ldi r17, low(RAMEND)               ;initialising stack pointer
out SPL , r17                      ;needed for calling any subroutine
ldi r17 , high(RAMEND)             ;loading stackpointer to ram end adress
out SPH , r17

ldi uni, 0xFF//initialise portb and portd as outports
out ddrd, uni
out ddrb, uni

ldi reg17, 0x38
rcall LCDCommand
ldi reg17, 0x0E
rcall LCDCommand
ldi reg17, 0x06
rcall LCDCommand
ldi reg17, 0x01
rcall LCDCommand
ldi reg17, 0x80
rcall LCDCommand

ldi r23,1                          ;counter register
```

```
sei                                          ;enable global interrupts
clr r16
ldi r16,(1<<WGM01)+(1<<CS00)+(1<<CS01)
                                             ;WGM01 for CTC and CS00/01 for prescaling
out TCCR0 , r16                              ;prescaler 64
ldi r16 , 2                                  ;compare match interrupt
out TIMSK , r16
ldi r16 , 50
out OCR0 , r16
sbr r17, 1<<ADLAR + 1<<REFS0                 ;set reference to AVcc by REFS0 and
                                             ;get a 8bit result on ADCH by left adjusting via
ADLAR
out ADMUX, r17
sbr r16, 1<<ADPS2 + 1<<ADPS1 + 1<<ADPS0 + 1<<ADIE + 1<<ADEN
                                             ;prescaler - 16MHz/128
                                             ;1<<5 for free running mode, ADIE - adc
interrupt enable
                                             ;ADEN- adc enable
sbr r16, (1<<ADSC)                           ;start conversion
out ADCSRA, r16
loop:
rjmp loop

timer0_ISR:
inc r23                                      ;increment counter
cpi r23,100
breq sqrt
clr r16
sbr r16, 1<<ADPS0 + 1<<ADPS1 + 1<<ADPS2 + 1<<ADEN + 1<<ADIE + 1<<ADSC
                                             ;start next conversion
reti

ADC_ISR:
in r16, ADCH                                 ;get i/p from ADC conversion
rcall ADC_result_convert
reti                                         ;return from interrupt

;************************************************;
ADC_result_convert:

;Q3.5 of Vref = 5 x 2^5 = 160 = b10100000
;Q0.8 of ADC result is ADCH so to get 3.5 version
;Q0.8 x Q3.5 = Q3.13    Q3.13/2^8 = Q3.5

ldi r17,160
mul r16,r17
out portd,r1
;the result of mul is stored in r1:r0
```

```
;divide by 256 = 2^8 means shifting 8 times to the right
;other wise we can just ignore r0 register
;result in r1 register

rcall calc
ret
;*****************************************************;

calc:
mul r1,r1
add r19,r1
ldi r18,0
adc r20,r18
ret

bcd_convert:
mov r25,r0
push r0
rcall divide10
ldi r26,48
add r24,r26                                    ;ASCII value of '0' = 48
mov r17,r24
rcall lcd_display
ldi r25,10
mul r25,r1
mov r25,r0
pop r0
sub r0,r25
add r0,r26                                     ;ASCII value of '0' = 48
mov r17,r24
rcall lcd_display
ret


;*********************************************************;
divide10:

;loop execute divide by 10
;1/5 x 255 = 255 / 5 = 51 and 255 nearly equal to 256
;since we are dividing by 1
;A / 10 ~= [ [ ( A x 51 ) + 1 ] / 2 ] / 256
;256 x [A / 10] ~= ( 51A + 1) / 2
;256 x [A / 10] ~= 51A/2 + 1/2
;256 x [A / 10] ~= 51A/2
;256 x [A / 10] ~= 25.5 x A
;256 x [A / 10] ~= 25A
;since we dropped 1/2 earlier so we are multiplying by 26 instead of 25
;so  256x[A/10] ~=26A
```

```
;so we multiply with 26 and ignore lower byte of result

ldi r24,26
mul r24,r25
mov r24,r1
ret

sqrt:
ldi r23,1
CLR   REML                    ;Initialize Remainder to zero
CLR   REMH                    ;
CLR   ANSL                    ;Initialize Root to zero
CLR   ANSH                    ;
LDI   C,8                     ;Set Loop Counter to eight

here:

LSL   ANSL                    ;Multiply Root by two
ROL   ANSH                    ;
LSL   AL                      ;Shift two high-bits of Square
ROL   AH                      ;into Remainder
ROL   REML                    ;
ROL   REMH                    ;
LSL   AL                      ;Shift second high bit of Sqaure
ROL   AH                      ;into Remainder
ROL   REML                    ;
ROL   REMH                    ;
CP    ANSL,REML               ;Compare Root to Remainder
CPC   ANSH,REMH               ;
BRCC  SKIP                    ;If Remainder less or equal than Root
INC   ANSL                    ;Increment Root
SUB   REML,ANSL               ;Subtract Root from Remainder
SBC   REMH,ANSH               ;
INC   ANSL                    ;Increment Root

SKIP:

DEC   C                       ;Decrement Loop Counter
BRNE  here                    ;Check if all bits processed
LSR   ANSH                    ;Divide Root by two
ROR   ANSL                    ;
mov   r16,r0
cpi   r16,127
brne  signal
rcall bcd_convert
rcall delay
ret
```

```
delay:
ldi r16,255
here3:
ldi r17,10
here2:
dec r17
brne here2
dec r16
brne here3
ret

LCDCommand:
rcall ready
out PortB , reg17
cbi PortD  , L_RS
cbi PortD  , L_RW
sbi PortD  , L_E
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
cbi PortD, L_E
ret

LCD_display:
rcall ready
out PortB , reg17
sbi PortD , L_RS
cbi PortD , L_RW
sbi PortD , L_E
nop
nop
nop
nop
nop
```

```
nop
nop
nop
nop
nop
nop
nop
nop
nop
cbi PortD , L_E
ret

ready:
cbi PortD , L_E
ldi uni, 0x00
out ddrb, uni
cbi PortD , L_RS
sbi PortD , L_RW

wait:
cbi PortD, L_E
nop
nop
sbi PortD, L_E
sbic PinB, 7
rjmp wait
cbi PortD, L_E
ldi uni, 0xFF
out ddrb, uni
ret

signal:
ldi R17 , 255
out PORTC , R17
ret
END
```